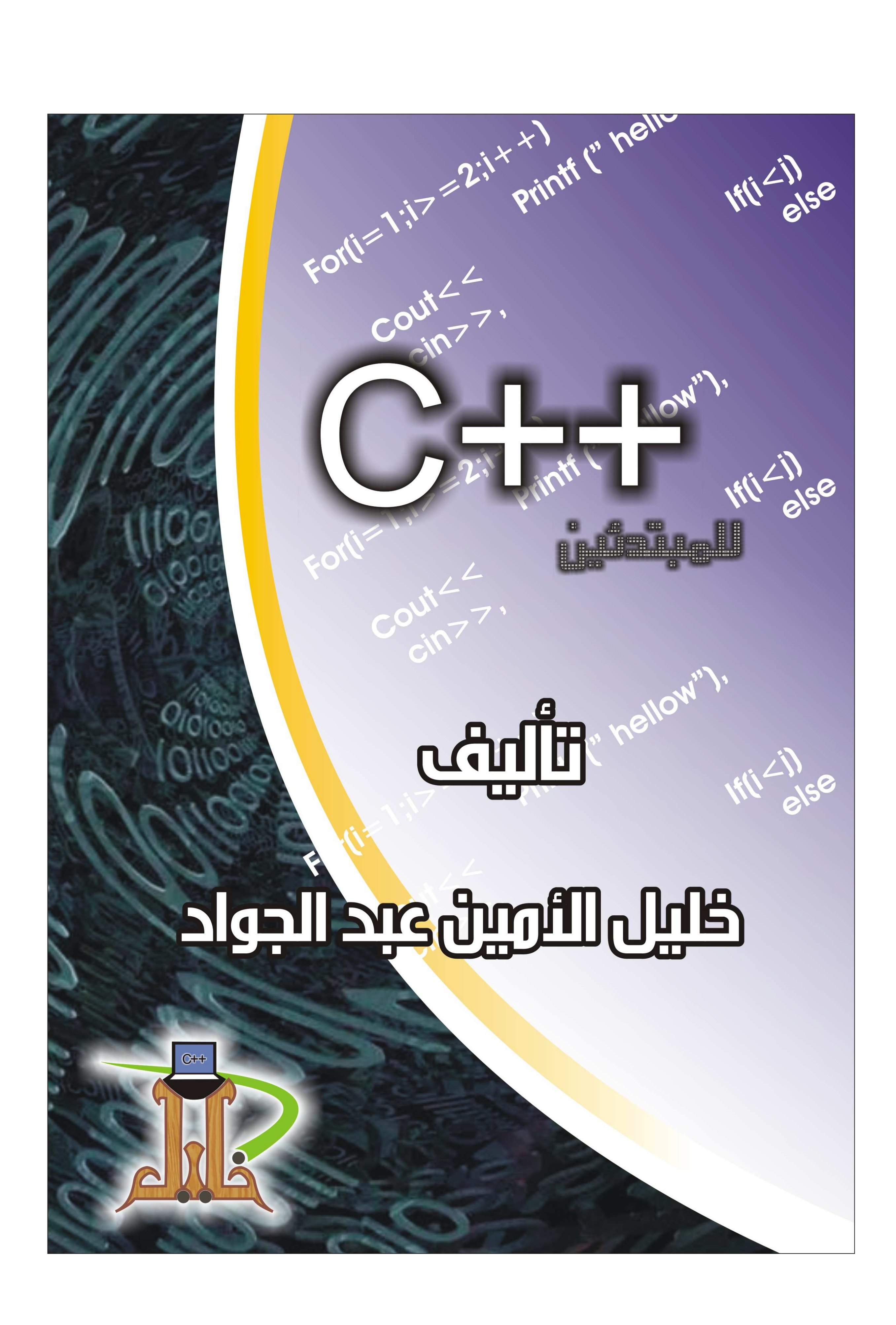


C++
للمبتدئين
تأليف
خليل الأمين عبد الجواد


بسم الله الرحمن الرحيم

مقدمة

قمت بكتابة هذا الكتاب راجياً أن يكون مفيداً لمن يدرس لغة سي ++ لأول مرة , إذ أني اتبعت فيه أسلوب التبسيط والتوضيح والشرح السهل لأهم أساسيات وركائز اللغة . والله أسأل أن أكون وفقت في ذلك .

وإذا كانت لديك أي ملاحظة أو تعليق على الكتاب فيمكن إرسالها إلي

بريدي الإلكتروني :

Khal_i_l@Yahoo.com

خليل الأمين عبد الجواد

طرابلس – ليبيا

2007/11/2

حروف اللغة

حروف لغة سي++ هي التالية :

- الأرقام العربية وهي 0,1,2,3,4,5,6,7,8,9,10 .
- الأحروف الهجائية الإنجليزية A,B,C,…,X,Y,Z و a,b,c,…,x,y,z .
- الرموز الخاصة مثل # & ^ % ! .
- ولغة سي++ حساسة لحالة الأحرف أي أن cout ليست نفسها Cout .

البرنامج الأول :

البرنامج التالي البسيط يبين تركيب البرنامج في لغة سي++

```
#include<iostream.h>
// My first c++ programming
main()
{
    cout<<"Welcome to C++";
return 0;
}
```

شرح البرنامج :

السطر الأول فيه يتم تضمين الملف iostream.h وهو مكتبة الإدخال والإخراج للغة سي++ , وذلك لأن أساليب الإخراج والإدخال غير مضمنة في اللغة , ولكنها موجودة في المكتبات المضمنة مع اللغة , ويتم التضمين باستخدام الأمر include أي ضمّن , ويسمى أي ملف ينتهي بالامتداد h. بالملف الرأسي Header file وهو يحتوي عادة على فئة وتراكيب بيانات دوال وثوابث , ويتم إنشاؤه عندما تكون هذه العناصر البرمجية عامة الاستخدام أي أنها ستستخدم في عدة برامج , ومن ثم بدل كتابتها كل مرة يتم كتابتها في ملف رأسي ثم تضمنيه كل مرة في البرنامج الذي نحتاج فيه لهذه التركيبات .

والأمر include مسبوقٌ بالرمز # وكل أمر يسبق بهذا الرمز يسمى موجه ما قبل المعالجة Preprocessor directive , أي أن مترجم اللغة يقوم بتنفيذ ما يمليه هذا الموجه قبل أن يقوم بترجمة البرنامج , فمثلاً في السطر الأول من البرنامج فإن المترجم يقوم بتضمين الملف iostream.h في البرنامج الحالي قبل أن يترجمه .

السطر الثاني يبدأ بالرمزين // , أي نص يأتي بعد هذين الرمزين إلى نهاية السطر يسمى تعليقاً , وهو نص يكتبه المبرمج متى أراد لكي يكتب معلومات عن البرنامج من التعريف بعمل البرنامج ومبرمجه ومتى تمت برمجته , كما يكتب لكي يشرح جمل البرنامج كيف عملها والغرض منها , وهذا مهم جداً لتطوير البرنامج , لأنه إن عدت إلى البرنامج بعد مدة وأردت تطويره ولم يكن فيه تعليقات موضِحة فإنك لن تفهم شيئاً منه والأغلب أنك لن تستطيع تطويره إلا إذا أعدت كتابته من جديد! , لذا فإن البرنامج ليكون مبرمجاً بطريقة جيدة لا بد أن يحتوي على تعليقات , ولا يعني هذا كتابة التعليقات في كل مكان فالغرض هو كتابة البرنامج لا التعليقات , وإنما تكتب لتوضيحه , ومن ثم ينبغي كتابتها في الأمكنة التي تحتاج إلى توضيح أما التي لا تحتاج فلا داعي للتعليق عليها .

السطر الثالث يحتوي على الدالة الرئيسية main وبعدها قوسان إذ كل دالة لا بد أن تتبع بقوسين , هذه الدالة منها يتم بدء تنفيذ أي برنامج بالسي++ لذا فإنه لا بد من وجودها , ويتم تنفيذ الجمل البرمجية المحتواة داخلها .

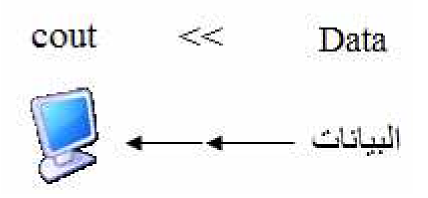
في السطر الرابع يوجد القوس المنبعج { والذي يعني بداية جسم الدالة الرئيسية .

السطر الخامس يبدأ بكلمة cout وهي اختصار للجملة Course output أي منهج الخرج والذي هو الشاشة في نظام Unix , و cout هو كائن يقوم بإخراج ما يأتي بعده على الشاشة ويسمى بنهر أو مجرى الإخراج.

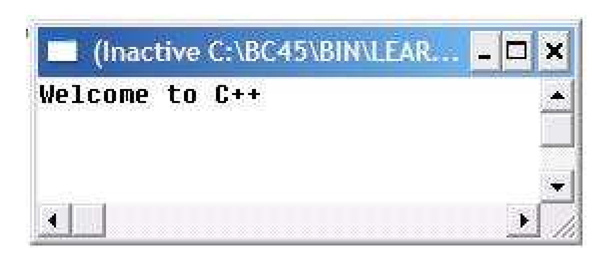
وهذا الكائن موجود ومعرف في المكتبة iostream لذلك تم تضمينها مسبقاً

ويكتب بعد cout علامتي أكبر من << ووهما معاً يكونان معاملاً يسمى معامل الإخراج والذي يقوم بإرسال ما يأتي بعده إلى الكائن cout .

ولحفظ اتجاه العلامتين فإننا نعتبر أن cout هي الشاشة وأن العلامتين هما سهمان يشيران إلى اتجاه البيانات كما في الشكل التالي :



بعد معامل الإخراج كتبت الجملة "Welcome to C++" وهي تبدأ بعلامة التنصيص المفردة ثم النص ثم علامة تنصيص أخرى , كل ما يكتب بين علامتي تنصيص فإنه يخرج مثلما هو على الشاشة , أي أن ناتج تنفيذ البرنامج هو التالي :



وجملة الإخراج السابقة قد انتهت بفاصلة منقوطة ; لأنه في لغة السي++ كل جملة برمجية يجب أن تنتهي بالفاصل المنقوطة , والجملة البرمجية هي أي جملة قائمة بذاتها وتقوم بعمل ما بنفسها ولا تعتمد على جملة تأتي بعدها .

ونسيان الفاصلة المنقوطة أكثر خطأٍ يقع فيه المبتدئون .

السطر قبل الأخير يحتوي على ;return 0 وهو تقوم بإنهاء البرنامج , والرقم صفر يعني انتهاء البرنامج بنجاح .

السطر الأخير يوجد فيه القوس } والذي يعني نهاية جسم الدالة الرئيسية.

والتركيبة السابقة ثابتة وضرورية في كل برنامج سي++.

عملية الإخراج :

كما مر بنا فإن الكائن cout هو المتحكم في إخراج البيانات على شاشة الحاسب , وهو كائن مرن جداً وفي الحقيقة فإن طريقته أفضل طريقة وجدتها في كل لغات البرمجة من حيث البساطة والمرونة .

ويمكن أن نقوم بإخراج أكثر من عنصر بيانات في المجرى الواحد باستخدام معامل الإخراج قبل كل عنصر نريد إخراجه كالتالي :

```
cout<<"first datum"<<" second datum";
```

وبالطبع هو لا يقوم بإخراج النصوص فقط ولكن الأعداد بكل أنواعها ونواتج العمليات الحسابية والمنطقية أيضاً.

فمثلاً لإخراج 40=8*5 على الشاشة يمكننا كتابتها كالتالي:

```
cout<<"5*8="<<5*8;
```

أو

```
cout<<5<<"*"<<8<<"="<<5*8;
```

ولغة السي++ تعطي حرية كبيرة في كتابة الكود بعدة أشكال وكيفما يريد المبرمج , من ثم يمكن كتابة جملة الإخراج في أكثر من سطر بحيث ينبغي أن يبتدئ كل سطر بمعامل الإخراج , أي يمكن كتابة الجملة السابقة كالتالي :

```
cout<<"5*8="
<<5*8;
```

ويمكن تنسيقها لتكون في صورة أفضل كالتالي :

```
cout<<"5*8="
      <<5*8;
```

وللذهاب إلى سطر جديد يتم استخدام الكلمة endl والتي هي اختصار end line أي نهاية السطر في أي مكان في جملة الإخراج كالتالي :

```
cout<<"first line"<<endl;
cout<<"second line";
```

ويمكن كتابتها هكذا:

```
cout<<"first line"<<endl<<"second line";
```

كما يوجد في اللغة بعض الرموز الحرفية الخاصة والتي تسمى بحروف الهروب Escape Characters وتقوم بوظائف معينة عند إخراج البيانات على الشاشة , وهي أحرف مفيدة للمبرمج , وهذه الحروف لا بد أن تكون مكتوبة بين علامتي تنصيص سواء أكانت بجانب نص

أو مفردة ,هي تتكون من رمزين أولهما الرمز \ حيث أن أي حرف أو رمز بعد هذا الرمز يعامل معاملة خاصة , ولإظهار الرمز \ على الشاشة تكتب جملة الإظهار كالتالي :

```
cout<<"\\";
```

والجدول التالي يبين بعض هذه الحروف :

الحرف	التأثير
\n	سطر جديد
\b	مسافة إلى الخلف
\t	7 فراغات أفقية
\r	الرجوع إلى بداية السطر
\a	الإنذار بالجرس
\'	لطباعة علامة التنصيص '
\"	لطباعة علامة التنصيص "
\?	لطباعة علامة الاستفهام ?

المتغيرات :

المتغيرات هي أسماء لمواقع في الذاكرة العشوائية RAM , هذه المواقع سيتم فيها تخزين البيانات حسب نوع المتغير , وهذه البيانات يتم التعامل معها في البرنامج لأداء المطلوب منه , وهذه البيانات قد تكون أعداداً صحيحة أو كسرية أو نصية أو صوراً أو أي نوع من البيانات التي يتعامل معها الحاسب .

وكل خلية من الذاكرة يعطيها نظام التشغيل عنواناً في هيئة النظام السداسي عشر شبيه بالتالي 0x270f222a, ومن المستحيل إذا أردت استخدام هذه الخلايا لتخزين البيانات فيها أن تقوم بحفظ عناوينها ولذا يتم استخدام المتغيرات لإعطاء الخلايا أسماء واضحة تسهل علينا التعامل مع الذاكرة , كما أن المتغيرات يمكن أن تكون تجميعاً لأكثر من خلية ذاكرة إذا لم تكن الخلية الواحدة كافية لحفظ قيمة المتغير.

والمتغيرات من أساسيات البرمجة , وكل برنامج حقيقي لا بد أن يحتوي عليها .

وفي لغة سي++ فإن كل متغيرٍ لابد من تعريفه والإعلان عنه أولاً قبل استخدامه .

قيمة المتغير :

هي القيمة التي سيتم تخزينها في الخلية أو الخلايا المعبر عنها باسم المتغير , وهي قيمة غير ثابتة بل يتم تغييرها حسب ما يريد المبرمج .

أنواع المتغيرات :

هناك أنواع للمتغيرات بحيث أن المتغير من النوع س يختلف عن المتغير من النوع ص من حيث نوع البيانات التي يستطيع التعامل معها والمدى الذي يمكن أن تصله هذه البيانات.

الأنواع الرئيسية :

النوع الصحيح Integer :

وهو النوع الذي يسمح بتخزين الأعداد الصحيحة فيه , والعدد يمكن أن يكون موجباً أو سالباً , ولتعريف متغير يتم كتابة كلمة int وهي الثلاثة أحرف الأولى من Integer وبعدها اسم المتغير المراد تعريفها كالتالي :

```
int number;
```

ولتعريف أكثر من متغير في جملة واحدة يتم الفصل بين أسماء المتغيرات بالفاصلة كالتالي :

```
int Day,size,ID;
```

التخصيص :

يتم تخصيص أو إسناد القيم وتخزينها في المتغيرات بكتابة اسم المتغير ثم معامل التخصيص = ثم القيمة المراد تخصيصها , فلتخصيص القيمة 10 للمتغير a والقيمة 5 للمتغير b نكتب التالي :

```
int a,b;
a=10;
b=5;
```

ويمكن كتابة جملتي التخصيص السابقتين معاً بشرط الفصل بينهما بالفاصلة كالتالي :

```
a=10,b=5;
```

وإذا أريد تخصيص القيمة 10 للمتغيرين فيمكن كتابة التالي:

```
a=b=10;
```

وهذا ما يسمى بالتخصيص المتسلسل .

والقيمة المخصصة يمكن أن تكون تعبيراً رياضياً وليس عدداً صريحاً كالتالي :

```
a=5*10/2+6;
```

حيث يتم حساب ناتج العملية الحسابية ثم تخصيصه للمتغير .

ويمكن أن يتواجد داخل التعبير الرياضي متغير مثل :

```
b=10+a;
```

ويمكن للتخصيص أن يكون متسلسلاً كالتالي :

```
b=(a=10)+5;
```

القيم الابتدائية :

يمكن أن تُخصص للمتغيرات قيم ابتدائية في جملة الإعلان عنها كالتالي :

```
int a=10,b=a+5;
```

ولا يمكن كتابة الجملة السابقة كالتالي:

```
int b=a+5, a=10;
```

وسيظهر المترجم رسالة خطأ وذلك لأن عملية التعريف تبدأ من اليسار ومن ثم فإن المتغير a سيعرف بعد المتغير b , ولذلك فعند إسناد قيمة a إلى b يكون المتغير a غير معرف .

وإذا تم إسناد قيمة كسرية للمتغير الصحيح فإن العدد الكسري سيتم حذفه ويتم تخزين القيمة الصحيحة فقط .

ولأن المتغير يشير إلى عنوان خلية في الذاكرة فإنه يمكن الحصول على هذا العنوان باستخدام معامل العنوان كالتالي:

```
cout<<&a;
```

وكل متغير له قيمة صغرى وقيمة عظمى من البيانات التي يتعامل معها ليقوم بتخزينها ولا يمكنه أن يخزن أكثر من القيمة العظمى ولا أقل من الصغرى , وإذا ما تم إسناد قيمة أكبر من القيمة العظمى أو أصغر من القيمة الصغرى – وهذا ما يسمى بالفائض الحسابي overflow – فإنه لن يتم تخزينها , وستُخزن بدلاً منها قيمة عشوائية أخرى .

وفي الحقيقة ما يحدث أنه إذا كانت القيمة أكبر من القيمة العظمى فإنه يتم الذهاب إلى القيمة الصغرى والزيادة منها بحسب القيمة المتبقية من طرح القيمة العظمى من القيمة المسندة .

ومدى القيمتين العظمى والصغر أو حجم المتغير يختلف من مترجم إلى آخر , وهو يقاس بالبايت Byte وهو للمتغير الصحيح عادة ما يكون إما 2 بايت أو 4 بايت وفي المترجم Borland C++ هو 2 بايت , ويمكن معرفة حجم المتغير باستخدام المعامل sizeof كالتالي:

```
cout<<sizeof(int);
```

أو بتعريف متغير ثم حساب حجمه كالتالي:

```
int a;
cout<<sizeof(a);
```

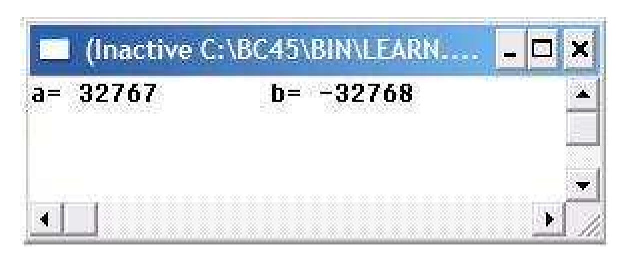
وإذا كان حجم المتغير الصحيح 2 بايت فإن قيمته القصوى هي 32767 وقيمته الصغرى -32768 .

وعند كتابة الكود التالي :

```
int a,b;
a=32767;
b=a+1;
```

```
cout<<"a= "<<a<<"      b= "<<b;
```

فالنتيجة هي التالية :



وهذا يبين أن قيمة المتغير تدور بين النهاتين العظمى والصغرى , الفائض الحسابي خطأ يحدث أثناء تنفيذ البرنامج وإذا حدث فإن البرنامج لن يعمل كما يراد له , والمشكلة الكبرى في هذا الخطأ أنه لا يمكن معرفة حدوثه وذلك لأنه لا يسبب في انهيار البرنامج وتوقفه بل يعمل البرنامج بشكل طبيعي ولكن النتائج لن تكون طبيعية طبعاً , ولذا يجب الحذر من الوقوع فيه .

النوع الصحيح الطويل Long :

وهو كالنوع int إلا أن قيمتيه العظمى والصغرى أكبر منه , ويتم تعريف المتغيرات منه كالتالي :

```
long a,b,c;
```

نوع الفاصلة العائمة float :

وهو يتعامل مع الأعداد الحقيقة أي الصحيحة والكسرية معاً , وتعريف المتغيرات من هذا النوع يأخذ الشكل التالي:

```
float var1,var2;
```

النوع الحقيقي المضعف Double :

وهو مثل النوع float إلا أنه يأخذ قيمة أكبر وذو دقة أكبر أيضاً , وتعرف متغيراته كالتالي:

```
double speed,gravity;
```

النوع الحرفي Character :

وهو يقوم بتخزين الحروف الأبجدية والأرقام من 1 إلى 9 والرموز الخاصة مثل @ , ! ويعرف كالتالي :

```
char a,b,c;
```

ويسند إليه الحرف بوضعه بين علامتي تنصيص مفردتين كالتالي:

```
a='A',b='7',c='?';
```

وفي الحقيقة فإن المتغير الحرفي أحد أنواع المتغيرات الصحيحة ؛ وذلك لأن قيمة المتغير يتم تخزينها بشفرة آسكي ASCII code والتي هي عبارة عن ارقام صحيحة , حيث يمثل كل حرف بأحد هذه الأرقام , وعند إرسال الحرف إلى مجرى الإخراج فإنه يتم إرسال القيمة المقابلة لقيمة آسكي .

فمثلاً الحرف A تقابله القيمة 65 في شفرة آسكي ويمكن بدلاً من تخصيص القيمة A إلى المتغير تخصيص القيمة 65 بدلاً منها كالتالي :

```
char c=65;
cout<<c;// will print A
```

ولإخراج قيمة آسكي المقابلة للحرف نكتب التالي :

```
cout<<int(c);
```

ولأن المتغيرات الحرفية هي متغيرات صحيحة فإنه بالإمكان تخصيص متغيرات حرفية إلى متغيرات من النوع int وبالعكس .

التحويل بين أنواع المتغيرات :

يمكن للمترجم التحويل بين أنواع المتغيرات إذا توجب ذلك , فمثلاً في التخصيص التالي :

```
int a=5;
float b=a;
```

فإنه قبل تخصيص المتغير a إلى b يتم تحويل وترقية المتغير a إلى النوع float ثم يتم تخصيصه , وهذا التحويل يسمى بالتحويل التلقائي وذلك لأن المترجم يقوم به تلقائياً , والشرط لحصوله أن يكون حجم المتغير المُخصص أصغر من المخصص إليه , وحجم النوع float أكبر من int .

أما إن أريد تخصيص متغير ذي حجم أكبر إلى متغير ذي حجمٍ أصغر فإنه يتم استخدام التحويل القسري , وصغته كالتالي :

```
var1=type(var2);
```

أو كالتالي :

```
var1=(type)var2;
```

حيث type هو نوع المتغير var1 والمتغير var2 هو المتغير ذو الحجم الأكبر . فمثلاً لتخصيص قيمة متغير حقيقي إلى متغير صحيح يتم ذلك بمثل التالي :

```
float a=100;
int b=int(a);
```

شروط تسمية المتغيرات :

هناك ضوابط لاختيار اسم المتغير وهي التالية :

- أن لا يبتدئ برقم .
- أن لا يحتوي على الرموز الخاصة باستثناء الرمز الشرطة السفلية under score _ .
- أن لا يكون من الكلمات المحجوزة في اللغة , وهي الكلمات التي من خصائص اللغة ولها معانٍ خاصة فيها .

والكلمات المحجوزة هي:

auto break case catch char class const continue default delete do double else enum extern float for friend goto if int inline long new operator private protected public register return short signed sizeof static struct switch template this throw typdef union unsigned virtual void volatile while

الثوابت :

الثوابت مثل المتغيرات إلا أنها لايمكن تغيير قيمتها والتي تخصص لها عند تعريفها مباشرة, وتعريفها مثل تعريف المتغيرات مسبوقاً بالكلمة const , مثل التالي :

```
const int a=100;
const float pi=3.14;
```

التعبير Expression :

يتكون التعبير من متغيرات أوثوابت أعداد أو نصوص يتم الربط بينها بالمؤثرات .

المؤثرات أو المعاملات Operators :

المعاملات هي رموز خاصة تقوم بعمل معين ولها عدة أنواع .

المؤثرات الحسابية :

و هي التي تقوم بتنفيذ العمليات الحسابية المعتادة مثل الجمع والطرح , و هي كالتالي :

المؤثر	معناه
+	الجمع
-	الطرح
*	الضرب
/	القسمة
%	باقي القسمة

كل المعاملات السابقة تستخدم مع الأعداد الصحيحة والحقيقية على السواء باستثناء معامل باقي القسمة فإنه يستعمل مع الأعداد الصحيحة والذي ينتج عنه باقي القسمة حينما يكون ناتج القسمة عدداً كسرياً , فمثلاً عند قسمة 5 على 2 كالتالي 5/2 فإن الناتج هو 2 وذلك لأن العددين صحيحين ومن ثم فالناتج عدد صحيح , أما باقي القسمة فهو 1 وذلك لأن 2*2=4 ويبقى 1 للوصول إلى 5 , ويمكن معرفة الباقي باستخدام المعادلة التالية باعتبار أن a , b عددين صحيحين :

a%b=a-(a/b)*b

والتعبير الرياضي يتم حسابه حسب قاعدة الأولويات للمؤثرات الحسابية حيث أن القوسين لهما الأولوية الأعلى وبعدهما معاملات الضرب والقسمة وباقي القسمة من اليسار إلى اليمين ثم معاملي الجمع والطرح من اليسار إلى اليمين , فمثلاً التعبير :

5+6*7/(2.5+4)

يتم حسابه كالتالي :

5+6*7/6.5

5+42/6.5

```
5+6.4615
11.4615
```

وفي الحقيقة فإن كل معامل في اللغة له أولوية محددة وليس المعاملات الحسابية فقط.

المؤثرات الحسابية المركبة :

هي مؤثرات ناتجة من جمع المؤثرات الحسابية مع معامل التخصيص = , وهي تستعمل للاختصار في الكتابة وهي : =* =/ += -= %= .

فمثلاً الكود التالي :

```
int i = 10;
i += 5 ;
```

السطر الثاني يعني أضف 5 إلى قيمة i ثم قم بتخصيص الناتج إلى i , وهو مكافئ للسطر:

```
i = i + 5;
```

مؤثرات الزيادة والنقصان :

يستعملان لزيادة أو إنقاص المتغير العددي بمقدار واحد صحيح .

معامل الزيادة هو ++ , ومعامل النقصان هو -- , فمثلاً قيمة i بعد الكود التالي ستكون 11 :

```
int a = 10 ;
a++;
```

و a الآن ستعود 10 :

```
a--;
```

ومعاملات الزيادة والنقصان إما أن تكون بعدية كما في المثالين السابقين أو قبلية وذلك بأن يسبقا اسم المتغير كالتالي:

```
++a;
--a;
```

والفرق بينهما هو أنه إذا وجد متغير الزيادة البعدية في تعبير ما ولنفترض التعبير التالي :

```
int a,b=20;
a = b++;
```

فإن قيمة a ستكون 20 وذلك لأنه تم تخصيص قيمة b إلى a أولاً , ثم تم زيادة قيمة b بواحد صحيح لتصبح 21 .

أما لو كان المؤثرُ مؤثرَ زيادة قبلية كالتالي :

```
a = ++b;
```

فإن قيمة a الناتجة ستكون 21 وذلك لأنه يتم إضافة واحد إلى b لتصبح 21 ثم تخصيص قيمتها الناتجة إلى a .

وأما إن وجد المؤثر مع المتغير مفردين فلا فرق بينهما , أي لا فرق بين السطرين التاليين :

```
a++;
++a;
```

وما مضى يطبق على مؤثر النقصان .

المؤثرات العلائقية :

وهي التي تستعمل في العمليات المنطقية وتبين العلاقة بين القيم أو التعابير الموجودة على طرفيها , وفي اللغة يوجد ست معاملات علائقية وهي :

المؤثر	معناه
==	يساوي
!=	لا يساوي
>	أكبر من
>=	أكبر من أو يساوي
<	أصغر من
<=	أصغر من أو يساوي

ونتيجة أي تعبير يحتوي على مؤثر علائقي هي إما صحيح True أو خطأ False , وفي لغة سي++ تمثل True الناتجة من تعبير في مؤثر علائقي بالرقم 1 و False بالصفر , فمثلاً جملة الطباعة التالية ستطبع 1 :

```
cout<<(5>2);
```

ولا بد أن يكون التعبير العلائقي بين قوسين .

المؤثرات المنطقية :

هي مؤثرات تقوم بالربط بين تعبيرات أو عناصر منطقية لتجعلها كتعبير واحد ونتيجتها أيضاً إما True أو False , وهي التالية :

المؤثر و And && :

وتكون نتيجة التعبير صحيحة إذا كان كلا التعبيرين أو العنصرين اللذين يربط بينهما صحيحاً والجدول التالي يبين كيفية عمله بافتراض أن A و B عنصرين أو تعبيرين منطقيين :

A	B	A&&B
False	False	False
False	True	False
True	False	False
True	True	True

ففي الجملة التالية ستتم طباعة صفر لأن 5 ليست أكبر من 7 :

```
cout<<(10==10 && 5>7);
```

المؤثر أو || :

وفيه تكون نتيجة التعبير صحيحة إذا كان أحد العنصرين صحيحاً والجدول يوضح ذلك :

A	B	A||B
False	False	False
False	True	True
True	False	True
True	True	True

مؤثر النفي Not ! :

ووهو مؤثر أحادي أي يعمل على عنصر أو تعبير واحد وتكون النتيجة المنطقية صحيحة إذا كان التعبير خاطئاً وخطأً إذا كان التعبير صحيحاً كالتالي :

A	!A
False	True
True	False

والجملة التالية ستقوم بطباعة 1 :

```
cout<<( !0);
```

الإدخال :

فيما سبق كنا نخصص القيم للمتغيرات أثناء تصميم البرنامج وبالتالي فإن هذه القيم ثابتة أثناء تنفيذ البرنامج , وبالتالي لا يمكننا تغييرها أثناء تنفيذه وفي الحقيقة فإن البرامج لا بد أن تغير القيم وتستقبلها أثناء التنفيذ , فمثلاً إذا أردنا حساب المتوسط لمجموعة من القيم ليست بثابتة فليس من المنطقي أن نعدل في الكود البرمجي كل مرة تتغير فيها القيم و ولكن المنطقي أن نقوم بإدخال القيم للبرنامج أثناء تنفيذه .

يتم إدخال القيم للمتغيرات باستخدام مجرى الدخل cin يتبعه معامل الإدخال >> كالتالي :

```
int i;
float f;
char c;
cin>>i>>f>>c;
```

أمثلة :

ملاحظة : في الأمثلة لن أقوم بكتابة هيكل البرنامج كاملاً إنما سأكتب الكود الذي يكون في داخل الدالة الرئيسية وذلك للاختصار .

1 – سنكتب برنامجاً يطلب من المستخدم إدخال عدد صحيح ويقوم بطباعة مربع العدد .

```
int number;
cout<<"Enter number : ";
cin>>number;
cout<<number<<"^2 = ";
cout<<number*number;
```

2- برنـامج يقوم باستقبال عددين صـحيحين ثم يقوم بطباعـة نـاتج جمعهمـا وضـربهما وحاصل طرح الثاني من الأول وحاصل قسمة الأول على الثاني :

```
float no1,no2;
cout<<"Enter two numbers :";
cin>>no1>>no2;
cout<<no1<<" + "<<no2<<" = "<<no1+ no2<<endl;
cout<<no1<<" - "<<no2<<" = "<<no1 - no2<<endl;
cout<<no1<<" * "<<no2<<" = "<<no1 * no2<<endl;
cout<<no1<<" / "<<no2<<" = "<<no1 / no2<<endl;
```

3- برنامج يحسب مساحة ومحيط مستطيل :

```
float area,length,width, circumference;//مساحة , طول , عرض , محيط
cout<<"Enter the length : ";
cin>>length;
cout<<"Enter the width : ";
cin>>width;
cout<<"\nArea = "<<length*width<<endl;
cout<<"Circumference = "<<2*(length+width);
```

4 – برنامج يحل المعادلتين :

```
x+y=no1
x-y=no2
```

حيث أن المدخلات هي no1 و no2 والمخرجات هي x و y :

```
float x,y,no1,no2;
cout<<"Enter no1 : ";
cin>>no1;
cout<<"Enter no2 : ";
cin>>no2;
x=(no1+no2)/2;
y=x-no2;
cout<<"\nx = "<<x<<endl<<"y = "<<y;
```

يتم الحل بجمع المعادلتين لحذف y وإيجاد قيمة x ثم التعويض بقيمة x لإيجاد قيمة y .

أسئلة :

1 - ما الأخطاء في البرنامج التالي :

```
int a=10;
cout<<a*5
float b=14.5;
cin<<b;
```

2 – اكتب برنامجاً يستقبل عددا حقيقياً يعبر عن المسافة بالكيلومترات ويحولها إلى أميال ثم يقوم بطباعتها , مع العلم أن الميل الواحد = 1.60934 كيلومتر .

الشروط والاختيارات

ما تعلمناه فيما سبق لا يمكننا إلا من برمجة القليل الذي يمكن أن نفكر فيه , فإذا أدخلنا عددين فلا يمكننا أن نعرف أيهما الأكبر وأيهما الأصغر , ولا يمكننا من برمجة برنامج لحل معادلة من الدرجة الثانية وغير هذا كثير ,أي أن البرامج السابقة كانت تنفذ حسب تسلسل الجمل ولم يكن بإمكاننا تنفيذ جمل وعدم تنفيذ أخرى حسب الظروف , ولكن مع جمل الشرط أو الاختيار يمكننا ذلك , ومن ثم فإن الجمل الشرطية هي إحدى اساسيات البرمجة .

الجمل الشرطية تقوم بفحص حالةٍ ما ثم تنفيذ جمل معينة بناء على تحقق الشرط أو عدمه , فمثلاً الجملة التالية جملة شرطية : إذا استمتعتَ بالبرمجة فستكون محترفاً , وهو يشبه الشرط البرمجي : إذا كانت قيمة المتغير س أكبر من 10 فإن س تكون 10.

ويمكن القول أن الجمل الشرطية تعطي الكومبيوتر جرعة من التعقل لأنها تمكنه من معرفة أشياء بناء على الشروط وبالتالي يقوم بالعمل وفقاً لها .

جملة إذا If statement :

جملة إذا هي جملة الشرط الرئيسية في اللغة , وتركيبتها كالتالي :

إذا (الشرط)	if(condition)
جملة	statement

والشرط يكون عبارة عن تعبير منطقي ربما يحتوي على مؤثرات علائقية أو ومنطقية و ربما لا, فمثلاً (6>5) و (4==8) هي شروط .

مثال :

البرنامج التالي يستقبل عدداً ويرى إن كان أكبر من 10 .

```
int i;
cin>>i;
if(i>10)
    cout<<i<< " > 10";
```

وقد ذكرنا من قبل أن كل تعبير منطقي قيمته إما 1 أو صفر وأن الصفر هي False وأن كل رقم ما عدا الصفر هو True .

وإن ما يفعله المترجم مع الشرط هو أن يحدد القيمة الناتجة عن التعبير المنطقي ثم يختبرها , فمثلاً في البرنامج البرنامج السابق لنفترض أن قيمى I هي 15 , ولأنها أكبر من 10 فإن قيمة التعبير هي 1 , و ما يحدث أن الشرط يصبح هكذا :

```
if(1)
```

وهذا يبين لنا أن الشرط ليس بالضرورة أن يكون تعبيراً منطقياً , فمثلاً يمكننا كتابة الشرط التالي في أي برنامج , وهو شرط لن يتحقق أبداً :

```
if(0) cout<<"It's zero .";
```

أما الشرط التالي فهو متحقق دائماً :

```
if(4) cout<<"True always ";
```

وإذا أريد تنفيذ أكثر من جملة إذ تحقق الشرط فإنه يتم كتابتهم بين قوسين منبعجين ليكونوا ما يسمى بالكتلة البرمجية كما يلي :

```
if(Condition)
{
    Statement 1;
    Statement 2;
```

```
    ...............;
}
```

الكتلة البرمجية Block:

هي مجموعة جمل يتم تنفيذها معاً ويمكن أن تحتوي على كل عناصر اللغة , تبدأ الكتلة بالقوس } وتنتهي بـ { , وما بين القوسين يسمى بمجال الكتلة , وجسم الدالة هو أيضاً كتلة وذا فإن برنامج كل برنامج السي++ هو كتلة أيضاً, ويمكن أن تحتوي الكتلة على كتل أخرى , وإذا تم تعريف متغير في كتلة فإنه غير معرف خارج مجالها ويسمى هذا المتغير بالمتغير المحلي لهذه الكتلة , أما إذا عرف في مجال خارجي فسيكون معرفاً داخل المجالات الداخلية , والمثال التالي يبين هذا :

```
{ // first block
    int x=1;
    { // second block;
        int y=10;
        cout<<x; // صحيح لأن المتغير معرف
    }
    cout<<y; // خطأ لأن المتغير غير معرف في هذا المجال
}
```

المتغير العام :

هو متغير يكون معرفاً خارج الدالة main ويكون معرفاً في كل المجالات , ففي المثال التالي المتغير x متغير عام :

```
int x=10;
main()
{
    cout<<x;
    return 0;
```

}

جملة إذا – وإلا if – else statement :

في مثالنا السابق كان البرنامج يطبع رسالة إذا كانت قيمة i أكبر من 10 ولا قوم بشيئ إذا لم تكن , وهذا يجعل البرنامج قاصراً على العمل التام , وتكون جملة الشرط غير مرنة كما يراد , ولكن تركيبة إذا وإلا تعالج هذا القصور, وصورتها كالتالي :

if (condition)	إذا(شرط)
Statement	جملة
else	وإلا
Statement	جملة

أي أنه إذا تحقق الشرط فإن الجملة أو الجمل التي بعد if سيتم تنفيذها , وإذا لم يتحقق فإن الجملة أو الجمل التي بعد else هي التي سيتم تنفيذها .

وهذا المثال السابق بإذا وإلا :

```
int i;
cin>>i;
if(i>10)
      cout<<i<<" > 10";
else
      cout<<i<<" <= 10";
```

* في الشروط المركبة أي التي تستخدم المعاملات المنطقية && أو || فإنه لن يتم اختبار الجزء الثاني من الشرط إلا إذا توجب ذلك , وهذا ما يسمى بالقصر short-circting , فمثلاً الشرط (p&&q) لن يكون صحيحاً إذا كانت قيمة p غير صحيحة أي false ومن ثم لا يتم اختبار قيمة q , وكذلك في الشرط (p||q) لن يتم اختبار قيمة q إذا كانت قيمة p صحيحة True .

ويجب الاستفادة من القصر لأنه يمنع أحياناً من انهيار البرنامج .

فمثلاً لكي يكون عدد ما يقبل القسمة على عدد آخر فإن باقي القسمة يجب أن يكون صفراً ولوضع هذا الشرط في جملة برمجية فإذا افترضنا أن العدد a سنقسمه على العدد b فإن جملة الشرط الصحيحة لذلك هي :

```
if(b>0 && a%b==0)
```

أما لو بدلنا التعبيرين عن جانبي المعامل && كالتالي :

```
if(a%b==0 && b>0)
```

فإن البرنامج يصبح عرضة للانهيار, وذلك لأنه إذا كانت قيمة b تساوي الصفر فسيتم محاولة القسمة على الصفر وهي غير معرفة وتسبب في انهيار البرنامج أما في الشرط الأول فإن لم تكن قيمة b أكبر من صفر فلن يتم اختبار التعبير a%b==0

جمل إذا المتداخلة :

يمكن أن تتواجد الجمل الشرطية واحدة داخل الأخرى مثل التالي :

```
if(condition)
{
      statement;
      if(condition)
      statement;
}
else if(condition)
          statement;
        else
          statement;
```

أو بأي صورة أخرى , والمهم التنبه إلى أن كل else تتبع if الي قبلها مباشرة .

في مثالنا السابق إذا لم يكن i أكبر من 10 فسيتم طباعة رسالة تقول أنه أصغر من أو يساوي 10 , ولكننا إذا أردنا أن نعرف هل هو 10 أم أصغر منها فإننا نستخدم إذا المتداخلة كالتالي:

```
int i;
```

```
cin>>i;
if(i>10)
      cout<<i<<" > 10";
else if(i==10)
      cout<<i<<" = 10";
        else
            cout<<i<<" < 10";
```

أمثلة :

1- طباعة أكبر قيمة وجملة تبين ذلك من بين ثلاث قيم :

```
int i,j,k;
cout<<"Enter three numbers : ";
cin>>i>>j>>k;
if(i>=j && i>=k)
      cout<<i<<" is the largest";
else if(j>=i && j>=k)
      cout<<j<<" is the largest";
else
      cout<<k<<" is the largest";
```

يلاحظ في else الأخيرة أنه لم يكن بعدها شرط وذلك لأنه غير ضروري إذ أنه إن لم تكن i هي الأكبر ولا j فلا بد أنها k .

مثال :

2 – برنامج يعرف هل القيمة المدخلة زوجية أم فردية :

فكرة التعرف على العدد هي أن العدد الزوجي إذا قُسم على 2 فإنه لا يوجد باقي قسمة أي صفر أما العدد الفردي عند تقسيمه على 2 فلا بد أن يكون هناك باقٍ , والبرنامج سيكون على الصورة التالية :

```
int number;
cin>>number;
if(number%2==0)
      cout<<number<<" is even.";
else
      cout<<number<<" is odd.";
```

ويمكن كتابة الشرط هكذا : ((number%2)!) , وذلك لأنه لو أن العدد هو 4 فإن باقي قسمته على 2 هو صفر ثم يقوم بواسطة معامل النفي ! بقلب الصفر واحداً ليصبح الشرط متحققاً , ومن ثم فإن 4 عدد زوجي , وتم كتابة القوسين بعد معامل النفي لأن أولوية معامل النفي أعلى من أولوية معامل باقي القسمة .

2- ما هي وظيفة البرنامج التالي :

```
      int i;
      cout<<"Enter number : ";
      cin>>i;
      if(i=0)
            cout<<"The number equal zero.";
      else if(i>0)
            cout<<"The number is positive.";
            else
                  cout<<"The number is negative.";
```

المفترض أن البرنامج يقوم بالتعرف على العدد المدخل هل هو موجب أم سالب أم يساوي الصفر , ولكنه لن يفعل وسيقوم بطباعة أن العدد سالبٌ دائماً وإن لم يكن كذلك .

لماذا؟

لأن الشرط هو (i=0) وليس (i==0) , فالذي سيحدث أنه سيتم تخصيص الصفر للمتغير i ثم اختبارها , ولأنها صفر فلن يتم تنفيذ جملة الطباعة الأولى ولا الثانية , أي كأننا كتبنا if(0)

4 – برنامج يقوم بطباعة اسم اللون حسب الحرف المدخل , فمثلاً إن كان b أو B يطبع Blue :

```
char c;
cin>>c;
if(c=='b' || c=='B') cout<<"Blue";
else if(c=='g' || c=='G') cout<<"Green";
else if(c=='r' || c=='R') cout<<"Red";
else if(c=='y' || c=='Y') cout<<"Yellow";
```

كان بالإمكان كتابة البرنامج بدون استخدام else كالتالي:

```
if(c=='b' || c=='B') cout<<"Blue";
if(c=='g' || c=='G') cout<<"Green";
if(c=='r' || c=='R') cout<<"Red";
if(c=='y' || c=='Y') cout<<"Yellow";
```

ولكن الكود الأول أفضل وذلك لأنه إذا تحقق الشرط الأول فلن يتم اختبار الثلاثة أسطر التالية , وإذا لم يتحقق الأول وتحقق الثاني فلن يتم اختبار السطرين الباقيين , أما في الكود الثاني فإن تحقق الشرط الأول أم لم فإنه سيتم اختبار كل الشروط وهذا يبطئ من سرعة تنفيذ البرنامج .

جملة التحويل Switch statement :

في المثال السابق كتبنا 4 جمل if ولكن تخيل لو أننا قمنا بفحص 20 حرفاً , في هذه الحالة سيكون العمل مملاً والأفضل أن تقوم بنسخ الأسطر ثم لصقها! بالتأكيد أمزح وهذا ليس حلاً جذرياً وتخيّل لو أنك كنت تبرمج في محرر نصوص صحراوي لا ماء فيه ولا نبات! , أعني لا نسخ ولا لصق وما شابه ماذا ستفعل؟ , اللغة توفر حلاً أفضل باستخدام جملة التحويل وتركيبتها كالتالي:

حوِّل(تعبير أو متغير)

}

حالة ثابت:

جمل

اقطع

حالة ثابت:

جمل

اقطع

....

تلقائي:

جملة

{

```
switch(expression or variable)
{
case constant:
    statement;
    break;
case constant   :
    statement;
    break;
```

```
………….
default:
      statement;
}
```

حيث أن التعبير يمكن أن يكون حسابياً أو منطقياً المهم أنه سيتم حساب قيمته .

وما يحدث عند case أنه يتم اختبار الثابث هل يساوي قيمة التعبير أو المتغير الموجود عند switch فإن كان يساويه فإنه يحقق الجملة أو الجمل التي بعد الحالة وإلا فإنه يتم تفحص باقي الحالات , أما الحالة default فإنه يتم تنفيذ الجمل التي بعدها إذا لم تتحق الحالات السابقة وهي ليست أساسية في التركيب ويمكنك ألا تكتبها.

الكلمة break :

إذا تحققت الحالة فإنه بعد تنفيذ الجمل التابعة لها فإن break تقوم بالخروج من جملة التحويل , وهي ضرورية لأنه إن لم تكتب فحتى إن لم تتحقق الحالات التالية فإنه سيتم تنفيذ جملها , ومن هذا يتضح أن آخر جملة لا تحتاج إلى break .

وإذا أريد تنفيذ نفس الجملة أو الجمل في حالة إذا تحققت الحالة أ أو الحال ب فيمكن كتابة ذلك كالأتي :

```
case constant:
case constant:
      statements;
```

أمثلة :

برنامج الألوان :

```
char c;
cin>>c;
switch(c)
```

```
{
case 'b':
case 'B':
       cout<<"Blue";
      break;
case 'g':
case 'G':
       cout<<"Green";
      break;
case 'r':
case 'R':
       cout<<"Red";
      break;
case 'y':
case 'Y':
       cout<<"yellow";
}
```

2 – سنبرمج آلة حاسبة تستقبل عدداً ثم معاملاً حسابياً ثم عدداً آخر وتوجد النتيجة حسب ماهية المعامل الحسابي :

```
float i,j;
char op;
cout<<"Enter number and math operator and another number : ";
cin>>i>>op>>j;
swich(op)
{
```

```
case '+':
      cout<<i+j;
      break;
case '-':
      cout<<i-j;
      break;
case '*':
      cout<<i*j;
      break;
case '/':
      cout<<i/j;
      break;
case '%':
      cout<<int(i)%(int)j;
      break;
}
```

3- برنامج يطبع قائمة على الصورة :

Choose number to find :

1 – Volt .

2 – Current.

3 – Resistance.

Enter your choice :

بحيث إذا تم اختيار 1 يتم حساب الجهد بقانون أوم بعدما يطلب البرنامج من المستخدم إدخال قيمتي التيار والجهد , ومثل هذا للخيارين الآخرين .

البرنامج :

```
float V,I,R;
int choice;
cout<<"Choose number to find :\n";
cout<<"1 - Volt.\n2 - Currrnt.\n3 - Resistance.\n";
cout<<"Enter your choice : ";
cin>>choice;
switch(choice)
{
case 1 :
      cout<<"Enter Current value : ";
      cin>>I;
      cout<<"Enter Resistance value : ";
      cin>>R;
      V=I*R;
      cout<<"\n\n\t\t****************\n";
      cout<<"\t\t*   V = "<<V<<" V   *";
      break;
case 2 :
      cout<<"Enter Voltage value : ";
      cin>>V;
      cout<<"Enter Resistance value : ";
      cin>>R;
      I=V/R;
      cout<<"\n\n\t\t****************\n";
      cout<<"\t\t* I = "<<I<<" A    *";
```

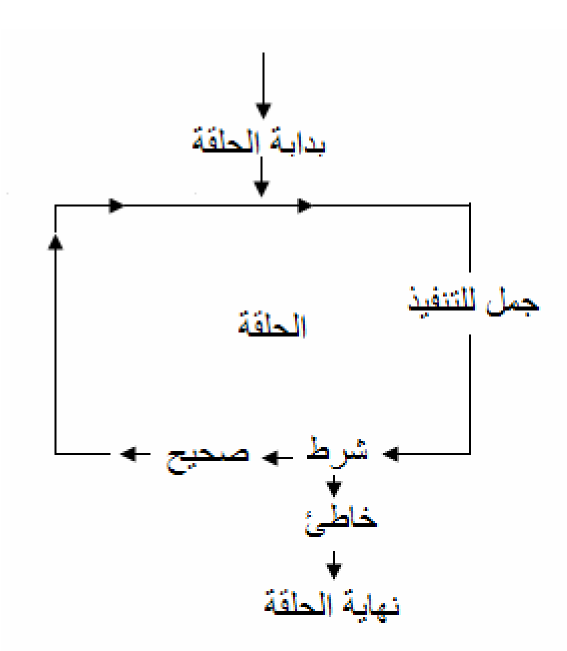
```
    break;
case 3 :
    cout<<"Enter Voltage value : ";
    cin>>V;
    cout<<"Enter Current value : ";
    cin>>I;
    R=V/I;
    cout<<"\n\n\t\t*****************\n";
    cout<<"\t\t* R = "<<R<<" Ohm     *";
    break;
}
    cout<<"\n\t\t*****************";
```

الحلقات التكرارية

رأينا كيف أننا باستخدام الشروط استطعنا من برمجة برامج أكثر من التي بدونها , ومع ذلك لا تزال البرامج الذي يمكننا برمجتها محدودة , ولا بد من وجود أشياء أخرى تمكننا من برمجة برامج أكبر وأكثر وأفضل , أول هذه الاشياء هي الحلقات التكرارية التي تقوم بتكرار تنفيذ جملة اوعدة جمل عدداً محدداً من المرات وهذا له فائدة قصوى والتكرار هو ثالث أساسيات البرمجة وهو مهم ولا يمكن تركه في برمجة أي برنامج وسنرى بعد أن ننتهي من شرح الحلقات والمصفوفات والدوال كيف يمكننا برمجة برامج ساحرة أساسها الحلقات .

مفهوم الحلقات :

الحلقة هي مجموعة من الأوامر يتم تنفيذها لأكثر من مرة حسب شرط محدد ويمكن توضيحها بالشكل التالي :



ويمكن أن لا يكون للحلقة شرط فتكون حلقةً لا نهائية أي يتم تكرارها عدد مرة غير محدود .

الحلقات التكرارية في اللغة :

يوجد في السي++ ثلاث أنواع من الحلقات هي :

1 – حلقة بينما While loop :

وتركيبتها كالتالي :

بينما (شرط)

}

الجمل المراد تكرارها;

مقدار الزيادة أو النقصان ;

{

```
while (condition)
{
statements;
increment or decrement value;
}
```

وتعني بينما الشرط صحيح كرر الجمل ومقدار الزيادة أو النقصان يقوم بزيادة أو إنقاص عداد الحلقة – والذي غالباً ما يكون في تعبير الشرط – حتى يتحقق الشرط .

مثال : البرنامج التالي يقوم بطباعة الأعداد من 1 إلى 10

```
int i=1;
while(i<=10)
{
     cout<<i<<"   ";
     i++;
}
```

ويمكن اختصار أسطر الحلقة هكذا :

```
while(i<=10)
     cout<<i++<<"   ";
```

أما لطباعة الأعداد الفردية فقط نعدل مقدار الزيادة ليكون i+=2; .

ولكي تكون الحلقة لا نهائية يمكن كتابة التالي :

```
while(1)
```

حلقة افعل-بينما do-while loop :

تركيبتها كالتالي :

```
do
{
statement;
increment or decrement value;
}
while(condition);
```

وهي تشبه حلقة while إلا أنه في هذه الحلقة إذا لم يتحقق الشرط مطلقاً فسيتم تنفيذ الجمل التي في داخل الحلقة مرة واحدة , أما في حلقة while فلن يتم تنفيذها مطلقاً .

حلقة لأجل for loop :

وهي الحلقة الأكثر مرونة واستعمالاً وتركيبتها كالتالي:

```
for(القيمة الابتدائية لعداد الحلقة;شرط التكرار;مقدار الزيادة أو النقصان)
{
statement;
}
```

فمثلاً مثال الأعداد من 1 إلى 10 السابق يكتب هكذا بـ for :

```
int i;
for(i=1;i<=10;i++)
      cout<<i;
```

وأفضل أن تقرأ الحلقة كالتالي:

i تساوي 1 ; بينما i أصغر من أو يساوي 10 ; i++ , وما أريده هو أن يتم قراءة الشرط كأنه شرط في الحلقة while , لأن قراءته وفهمه هكذا ستجعلك متمكناً أكثر من الحلقة for .

كما يمكن أن يكون التخصيص مع القيمة الابتدائية داخل قوسي الحلقة كالتالي:

```
for(int i=0;i<=10;i++)
```

كما يمكن أن يكون للحلقة أكثر من عدادٍ ويكون الشرط لكليهما أو أحدهما مثل التالي:

```
for(int i=1,j=-1;i<=10 && j>=-10;i++,j--)
```

كما يمكن أن يكون أحد عناصر الحلقةأو اثنان منهم أو كلهم ليس لهم وجود كالتالي:

```
int i=0;
for(;;)
cout<<i++;
```

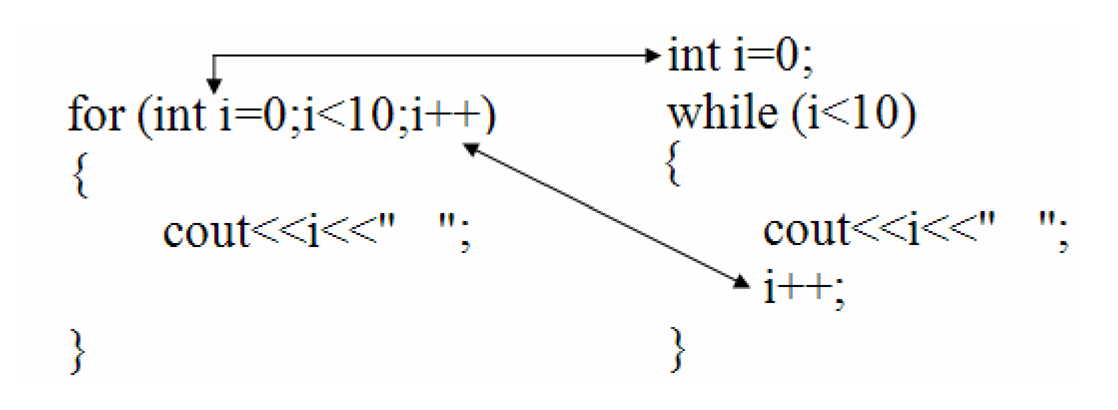
هذا الكود سيطبع ويزيد قيمة i إلى مالانهاية .

انظر إلى الكود التالي :

```
int i=1;
for(;i<=10;)
cout<<i++<<" ";
```

لقد قلت سبقاً أنه من الأفضل قراءة الشرط كأنه في حلقة while , انظر للتالي:

for(;i<=10;) == while(i<=10) وهي طريقة للتحويل بين for و while , أما الطريقة القياسية فهي التالية :



كلمتي continue و break :

تستخدم الكلمة المحجوزة continue للذهاب من مكانها إلى بداية الحلقة , وبالتالي لن يتم تنفيذ الكود المكتوب بعدها,أما break فهي تقوم بالخروج من الحلقة , والمثال التالي يوضح عملهما :

```
for(int i=0;i<10;i++)
{
    cout<<i<<"   ";
    if(i==5)break;
    continue;
    cout<<i*i;
}
```

في هذا المثال سيتم طباعة الأعداد من 1 إلى 5 , كما أنه لن يتم تنفيذ الجملة ;cout<<i*i .

أمثلة :

1 - مضروب العدد هو حاصل ضرب الأعداد من 1 إلى هذا العدد , فمثلاً مضروب الـ 5 هو 1*2*3*4*5 ,والبرنامج التالي يقوم بإيجاد المضروب لأي عدد صحيح :

```
int i;
long fact=1;
cout<<"Enter the number : ";
cin>>i;
for(int a=1;a<=i;a++)
    fact*=a;
cout<<"Factorial of  "<<i<<" = "<<fact;
```

بعد إدخال العدد يتم تعريف العداد a وتخصيص القيمة 1 كقيمة ابتدائية ويتم التكرار بينما قيمة العداد أصغر من أو تساوي العدد المراد إيجاد قيمته , مع زيادة العداد بمقدار 1 .

في الجملة التي في داخل الحلقة يتم ضرب قيمة المتغير fact في قيمة العداد a ثم تخصيصها إلى المتغير fact بحيث عند انتهاء الحلقة تكون قيمة fact هو المضروب المطلوب .

وعرفنا fact على أنه long لأن المتغير الصحيح int لا يستطيع احتواء أغلب قيم مضاريب الأعداد , فمثلاً لو كان حجمه 2 بايت فإن لا يمكن تخزين قيمة العدد 8 فيه.

2 – طباعة الأرقام في نفس المكان :

جملة الطباعة :

```
cout<<1<<"  "<<2;
```

تقوم بطباعة العددين بسرعة كأنهما كتبا معاً , ولكن إذا أردنا ان نؤخر البرنامج حين الطباعة فيمكننا استخدام الحلقة التالية :

```
long a,b=0;
for(a=1;a<10000000;a++)b+=a;
```

وذلك لأن الحلقة ستأخذ مدة من الزمن لكي تصل إلى 10000000 والطبع بزيادة هذه القيمة يزداد التأخير وبنقصانه يقل وتزداد سرعة الطباعة, وهذه الحلقة سنسميها حلقة التأخير , ومن ثم يمكن كتابة البرنامج لطباعة 2 بعد فترة من طباعة 1 كالتالي :

```
cout<<1<<"  ";
long a,b=0;
for(a=1;a<10000000;a++)b+=a;
cout<<2;
```

والآن لطباعة الأعداد من 1 إلى 100 في نفس المكان سنتخدم حرف الهروب \r الذي يقوم بإرجاع المؤشر إلى بداية السطر كالتالي:

```
for(int i=1 ;i<=100;i++)
   cout<<"\r"<<i;
```

عند تنفيذ السطرين السابقين سيتم طباعة الأرقام في نفس السطر , لأنه بعد طباعة أول قيمة سيعود المؤشر لموضع طباعتها وستكتب ثاني قيمة مكانها وهكذا , ولكننا لن نرى إلا العدد 100 من سرعة الطباعة , ولكي نرى كل الأعداد فإننا سنستخدم حلقة التأخير داخل الحلقة السابقة هكذا :

```
for(int i=1 ;i<=100;i++)
{
long a,b=0;
for(a=1;a<10000000;a++)b+=a;
```

```
cout<<"\r"<<i<<"%";
}
```

وتم إضافة الرمز % في جملة الطباعة ليظهر مثل ما يُرى في برامج التثبيت والفحص وغيرها .

3 – طباعة جدول آسكي :

```
for(int i=0;i<255;i++)
  cout<<i<<"  "<<char(i)<<"  ";
```

4 – الحلقة التالية تقوم بإظهار هرمٍ من النجوم :

```
for(int i=0;i<10;i++)
{
    int j;
   cout<<"                    ";
   for(j=0;j<=i;j++)cout<<"\b";
   cout<<'*';
   for(j=0;j<i;j++)cout<<'*';
   for(j=0;j<i;j++)cout<<'*';
   cout<<endl;
}
```

لتفهم كيفية عمله قم بجعل كلٍ من الحلقة الثانية والثالثة على هيئة تعليق ونفذ البرنامج لترى ما سيظهر , ثم انزع التعليق عن الثانية ونفذ البرنامج ثم عن الثالثة واستنتج طريقة العمل .

5 – البرنامج التالي يقوم بواسطة الحلقات بحركة رائعة , وهي أنه يحرك النص على الشاشة من اليسار إلى اليمين ثم من اليمين حتي يرجع إلى البداية :

```
for(int i=0;i<60;i++)
{
```

```
    long a,b=0;
    for(a=1;a<40000000;a++)b+=a;
    cout<<"\r";
    for(int j=0;j<=i;j++)
    cout<<" ";
    cout<<"We "<<char(3)<<" C++";
}
cout<<'\r';
for(int i=0;i<75;i++)
   cout<<" ";
for(int i=0;i<60;i++)
{
    long a,b=0;
    for(a=1;a<40000000;a++)b+=a;
    cout<<"\r";
    cout<<"                                                            ";
    for(int j=0;j<=i;j++)
    cout<<"\b";
    cout<<"We "<<char(3)<<" C++";
    cout<<"      ";
}
```

الدوال

معظم البرامج المفيدة أكبر بكثير من البرامج التي مرت بنا , لعمل برامج كبيرة يسهل تتبعها يقوم المبرمجون بتقسيم البرامج الرئيسية إلى برامج فرعية sub programs . هذه البرامج الفرعية في لغة سي++ تسمى دوال functions . يمكن ترجمة و اختبار البرامج الفرعية كل على حدة وإعادة استخدامها في برامج مختلفة .

وتنقسم الدوال إلى نوعين , دوال ترجع بقيمة ودوال فارغة .

التركيبة الشائعة لبرنامج يحتوي على دالة :

```
#include<iostream.h>
type function_name();// الإعلان عن الدالة
main()
{
function_name();// استدعاء الدالة
}
type function_name()  // تعريف الدالة
{
  // جسم الدالة //
}
```

حيث يستعمل الإعلان لتعرف الدالة main ان هناك دالة بهذا الاسم يجب البحث عنها , وهو يستعمل إن كان جسم الدالة مكتوباً بعد الدالة الرئيسية , أما إن كان قبلها فسيكون زيادة لا فائدة منها , كما يمكن أن يكون التعريف داخل الدالة الرئيسية قبل استدعائها , ولكن لا يفضَل هذا حتى يكون الكود واضحاً , ويتضح في التعريف أن اسم الدالة يكون مسبوقاً بالنوع وكذلك في تعريفها , ونوع الدالة إما يكون أحد الأنواع السابقة للمتغيرات وهذا إذا كانت الدالة ترجع بقيمة , أما إن لم تكن فإن النوع سيكون void أي فارغ , وإن لم يكتب نوع الدالة قبل استدعائها فستكون دالة من النوع int .

والدالـة يمكـن أن تسـتقبل بيانـات مـن مكـان اسـتدعائها , هـذه البيانـات تسـمى بالوسـائط أو البارامترات , وتوضع هذه الوسائط في قوسي الدالة كالتالي :

```
function(a,b);
```

حيث a و b هما الوسائط .

الدوال التي ترجع بقيمة :

وهـي التـي عنـد اسـتدعائها تقـوم بإرجـاع قيمـة إلـى مكـان الاسـتدعاء , فمـثلاً الدالـة التاليـة تقـوم بإرجاع مربع العدد المرسل إليها:

```
int square(int x);
main()
{
int i;
cin>>i;
cout<<square(i);
return 0;
}
int square(int x)
{
return x*x;
}
```

حيث الكلمة return هي التـي تقـوم بإرجـاع القميـة التـي تـأتي بعـدها , ونـوع الدالـة هـو نـوع القيمـة التـي تقـوم الدالـة بإرجاعهـا وهـي فـي المثـال السـابق integer واسـم الدالـة square , وتسـتقبل وسيطاً واحداً من النوع الصحيح .

ويظهر أنه سطر الإعلان عن الدالة ينتهي بفاصلة منقوطة , أما سطر بداية تعريفها فلا .

مثال : دالة تقوم بإرجاع أكبر قيمة من قيمتين :

```
float max(float x,float y)
```

```
{
    if(x>y)
       return x;
    else
       return y;
}
```

ويمكن استدعاء الدالة لإيجاد أكبر قيمة بين أكثر من قيمتين وكل القيم موجبة كالتالي :

```
main()
{
    cout<<"Enter number of values : ";
    int n;
    cin>>n;
    int max_value=0;
    for(int i=1;i<=n;i++)
    {
        int val;
        cout<<"Enter the value "<<i<<" : ";
        cin>>val;
        max_value=max(val,max_value);
    }
    cout<<max_value;
    return 0;
}
```

الدالة الفارغة void function :

هي دالة لا تقوم بإرجاع أي قيمة ولذا فيكون نوعها void , ومن استخداماتها أنه إذا كان ثمة عدة جمل متشابهة يراد كتابتها في الدالة الرئيسية في أكثر من مكان فإنه بدل كتابتها عدة مرات يتم كتابتها في دالة فارغة ثم يتم استدعاء الدالة بكتابة اسمها فقط في المكان الذي يراد كتابة الجمل فيه , والمثال التالي يبين دالة من هذا النوع تقوم بطباعة مكعب القيمة المرسلة إليها :

```
void cube(float x)
{
    cout<<x*x*x<<endl;
}
main()
{
    for(int i=1;i<10;i++)
    {
      cout<<i<<"^3 = ";
      cube(i);
     }
    return 0;
}
```

وكما قلنا في شرح الكتلة البرمجية فإن أي متغير معرف في دالة غير معرف في دالة أخرى .

الدوال المبنية في اللغة :

تتضمن اللغات عدة مكتبات تحتوي على دوال جاهزة ما على المبرمج إلا تضمين مكتباتها ثم استدعائها لتنفيذالمطلوب . ومثالٌ على هذه الدوال الدوالُ الرياضية .

ولا استخدام الدوال الرياضية يجب تضمني الملف math.h في بداية البرنامج , ومن الدوال الرياضية الموجودة :

الدالة	الوصف

acos(x)	معكوس جيب التمام
asin(x)	معكوس الجيب
atan(x)	معكوس الظل
ceil(x)	تقريب x لأصغر قيمة صحيحة أكبر من x
cos(x)	جيب التمام
exp(x)	رفع x للأساس e
fabs(x)	القيمة المطلقة
floor(x)	تقريب x لأكبر قيمة صحيحة أصغر من x
log(x)	اللوغاريثم الطبيعي
log10(x)	اللوغاريثم للأساس 10
pow(x,y)	x^y
sin(x)	الجيب
sqrt(x)	الجذر التربيعي
tan(x)	الظل

المثال التالي يقوم بطباعة الجيب والجذر التربيعي واللوغاريثم الطبيعي لعدد يتم إدخاله :

```
#include<iostream.h>
#include<math.h>
main()
{
    const double Pi=3.141592654;
    double x;
    cin>>x;
    cout<<"Sin "<<x<<" = "<<sin(x*Pi/180)<<endl;
    cout<<"Square root of "<<x<<" = "<<sqrt(x)<<endl;
```

```
    cout<<"ln "<<x<<" = "<<log(x);
    return 0;
}
```

تم ضرب الزاوية في ط وقسمتها على 180 عند إيجاد الجيب للتحويل من النظام الدائري إلى الدرجات .

الملفات الرأسية :

يمكننا أن نقوم بكتابة ملفات رأسية تحتوي على دوال نكتبها شائعة الاستعمال , بحيث عند الحاجة إليها في برامجنا نقوم بتضمين الملف ثم استدعائها , والملف الرأسي هو ملف نصي بسيط ينتهي بالامتداد h. يحتوي على الكود الذي نريده والذي غالباً ما يكون دوالَ أو ثوابت .
وعند تضمينه فإننا ننسخه إلى المجلد الذي يحتوي على ملفات البرنامج الذي نريد تضمينه فيه , ثم في الملف الرئيسي للبرنامج نكتب سطر التضمين كالتالي:

```
#include "file_name.h"
```

وتمت كتابة اسم الملف بين علامتي تنصيص لأنه معرف من قبل المبرمج أما الملفات التي تأتي مع اللغة فيتم تضمينها كما ضمنا iostream.h من قبل .
والمثال التالي يبين التعامل مع الملفات الرأسية :
في الملف الرأسي سنكتب دالتين , الأولى اسمها ()delay وهي تقوم بالتأخير باستخدام حلقة التأخير , والدالة الثانية اسمها ()type تقوم بطباعة الحروف الكبيرة ثم الصغيرة حرفاً حرفاً , أي أنها تستخدم الدالة ()delay , ومن ثم الملف الرأسي سيكتب فيه التالي:

```
#include<iostream.h>
void delay()
{
   long a,b=0;
   for(a=1;a<40000000;a++)b+=a;
```

```
}
void type()
{
   for(char i='A';i<='z';i++)
   {
      if(i>'Z' && i<'a')continue;
      if(i=='a')cout<<endl;
      delay();
      cout<<i<<" ";
   }
}
```

ثم نقوم بحفظه بـأس اسـم ولـيكن our_finctions.h , ثـم نقـوم بكتـابـة ملـف البرنـامج الرئيسـي , ويكون فيه التالي:

```
#include <iostream.h>
#include"our_functions.h"
main()
{
     type();
     return 0;
}
```

ويجب أن يكون الملفان في نفس المجلد .

الآن بعد تعرفنا على الدوال فإنه علينا أن نغير تفكيرنا ونظرتنا البرمجية , بحيث إذا أردنا أن نبرمج برنامجاً أن نفكر في الدوال وهل يحتاج البرنامج إلى كتابة دوال أم لا , وبصفة عامة فإن أهم أسباب كتابة الدوال هي التالية :

1 – إذا كان البرنامج كبيراً ومتشعباً بحيث إذا تمت برمجته ككتلة واحدة تصعب السيطرة عليه ويصعب تطويره .

2 – إذا وجد في البرنامج جمل برمجية ستتكرر كثيراً , ومن ثم بدل كتابها كل مرة يتم فصلها في دوال , ثم استدعاء هذه الدوال .

3 – إذا كان في البرنامج جزءًا يمكن أن يُكتب في برامج أخر , أي أن هذا الجزء عام , ومن ثم تتم كتابته في دوال وفي ملف رأسي منفصل .

أمثلة :

1 - دالة تقوم بإيجاد أكبر قيمة من 4 قيم بحيث تستخدم الدالة max() التي توجد أكبر قيمة من قيمتين والمذكورة سابقاً :

```
int max4(int a,int b,int c,int d)
{
    return max(max(a,b),max(c,d));
}
```

2 – دالة تقوم باختبار الحرف المرسل إليها وإرجاع قيمة صحيحة تدل على حالته إن كان رقماً أم حرفاً كبيراً أم حرفاً صغيراً :

```
int WhatIsChar(char c)
{
    if(c>='0' && c<='9') return 0;
    else if(c>='A' && c<='Z') return 1;
    else if(c>='a' &&c<='z') return 2;
}
main()
```

```
{
    while(1)
  {
    char c;
    cout<<"Enter char , to end enter '.' : ";
    cin>>c;
    if(WhatIsChar(c)==0)
      cout<<"It is digit."<<endl;
    else if(WhatIsChar(c)==1)
      cout<<"It is capital letter."<<endl;
    else if(WhatIsChar(c)==2)
      cout<<"It is smal letter."<<endl;
    else if(c=='.')break;
  }
}
```

3 – لقد علمنا أن الدالة sqrt() موجودة في الملف الرأسي math.h , ولكننا سنبرمج نفس الدالة من الصفر .

إحدى طرق إيجاد الجذر التربيعي تتم بتكرار المعادلة التالية :

$X_{i+1}=X_i^2+n/2*X_i$

حيث X هي الجذر التربيعي - وفي بداية تطبيق المعادلة تعطى القيمة 1 - للعدد n بعد تكرار المعادلة عدة مرات , ولإيجاد الجذر التربيعي لـ 4 يتم تكرارها 4 مرات , وكلما كبر العدد يزداد التكرار , ونحن سنكررها 15 مرة , والقيمة الناتجة تكون ذات دقة مقبولة .

والدالة هي :

```
double sqrt(double n)
{
```

```
    double x=1;
    for(int i=1;i<=15;i++)
     x=(x*x+n)/(2*x);
    return x;
}
```

الصفوف

Arrays

الصف هو تتابع من المتغيرات كلها من نفس النوع , هذه المتغيرات تسمى عناصر الصف ويتم ترقيمها بالتتابع 0 , 1 , 2 , ... هذه الأرقام تسمى الفهرس index أو الدلائل subscripts للصف , هذه الأرقام تحدد مكان العنصر في الصف.

والصف هو مصفوفة أحادية البعد .

وإذا كان اسم الصف ar ويحتوي على عدد n من العناصر فإن أسماء هذه العناصر ستكون ar[0] , ar[1] , ar[2] , ... , ar[n-1] , وإذا كان عدد العناصر n هو 5 فيمكن تصور الصف كالتالي :

ar	13	6	20	70	45
	0	1	2	3	4

حيث يحتوي العنصر الأول ar[0] على القيمة 13 والعنصر ar[1] على القيمة 6 .

وتعريف الصفوف يأخذ الشكل التالي:

```
type array_name[n];
```

حيث السطر التالي يعرف مصفوفة عناصرها من النوع الصحيح تحتوي على 10 عناصر :

```
int array[10];
```

ويفضل أحياناً تعريف حجم المصفوفة كثابت كالتالي:

```
const size=10;
int array[size];
```

ويمكن تخصيص قيم ابتدائية للمصفوفة كالتالي :

```
int array[10]={0,4,8,2,7,2,3};
```

حيث ستخصص القيم حسب ترتيبها للسبعة العناصر الأولى , أما باقي العناصر فستخصص لها القيمة 0 .

ولإسناد قيمة لعنصر يتم ذلك كالتالي:

```
array[4]=14;
```

حيث ستخصص القيمة 14 للعنصر الخامس في المصفوفة , ولتخصيص قيم لكل العناصر فبدلاً من كتابة السطر السابق لكل العناصر يتم ذلك بصورة أفضل باستخدام الحلقات مثل التالي:

```
for(int i=0;i<10;i++)
cin>>array[i];
```

وتتم طباعة كل عناصر المصفوفة باستبدال <<cin بـ >>cout .

أمثلة :

1 – مصفوفة حجمها 50 عنصراً , نريد إدخال بعض عناصرها أو كلها , ثم نطبع عناصر المصفوفة وأكبر قيمة بين عناصرها , وسيتم إدخال عدد العناصر المراد إدخالها أولا , وسيتم استخدام الدالة max() المذكورة سابقاً :

```
#include<iostream.h>
int max(int x,int y)
{
    if(x>y)return x;
    else return y;
}
main()
{
    int array[50],n,max_value,i;
    cout<<"Enter number of elements : ";
    cin>>n;
    for(i=0;i<n;i++)
{
    cout<<"Enter the element "<<i+1<<" : ";
    cin>>array[i];
}
```

```
	max_value=array[0];
	for(i=1;i<n;i++)
		max_value=max(max_value,array[i]);
	cout<<"The array is : ";
	for(i=1;i<n;i++)
		cout<<array[i]<<" ";
	cout<<"\nThe largest number is: "<<max_value;
return 0;
}
```

2 – إدخال مصفوفة صحيحة وطباعة عدد الأعداد الموجبة وعدد الأعداد السالبة وعدد الأصفار فيها :

```
int matrix[10],i,positive=0,negative=0,zero=0;
for(i=0;i<10;i++)
	cin>>matrix[i];
for(i=0;i<10;i++)
{
	if(matrix[i]>0) positive++;
	else if(matrix[i]<0) negative++;
	else zero++;
}
cout<<"Number of positive numbers is "<<positive<<endl;
cout<<"Number of negative numbers is "<<negative<<endl;
cout<<"Number of zeros is "<<zero<<endl;
```

3 – ترتيب عناصر مصفوفة ترتيباً تصاعدياً :

هناك عدة طرق للترتيب والطريقة التالية إحداها :

```
main()
{
   const int n=5;
   int l,i,j,k,temp,arr[n];
   for(i=0;i<n;i++)
   cin>>arr[i];
   for(i=0;i<n;i++)
   {
      temp=100;
     for(j=i;j<n;j++)
        if(temp>arr[j]){temp=arr[j],k=j;}
     arr[k]=arr[i];
     arr[i]=temp;
    for(l=0;l<n;l++)cout<<"   "<<arr[l];
    cout<<endl;
   }
   cout<<"\n\nThe array after sorting : ";
   for(i=0;i<n;i++)cout<<"   "<<arr[i];
}
```

الشرح:

في بعد إدخال البيانات يتم الدخول في حلقة for الثانية وفي أولها يتم إسناد القيمة 100 إلى المتغير المؤقت temp بحيث تم اعتبارها أنها أكبر قيمة يمكن أن تحتوي عليها المصفوفة .

ويتم استخدام المتغير temp لتبديل قيمتي متغيرين بحيث يحتوي كل متغير على قيمة المتغير الثاني , فمثلاً لا يمكن التبديل بين قيمتي المتغيرين a و b بالكود التالي:

```
a=b;
```

```
b=a;
```

وذلك لأن في الجملة الأولى سيأخذ a قيمة b أي ان قيمته الأولى ستمحى , وفي السطر الثاني فإن قيمة a ستخصص لـ b ولكن قيمة a هي قيمة b , أي لا فرق بين السطر الثاني والسطر :

```
b=b;
```

ولكن باستخدام متغير مؤقت يتم فيه تخزين قيمة b أولاً ثم يتم تخصيص قيمة a إلى b وتخصيص قيمة temp إلى a كالتالي:

```
temp=b;
b=a;
a=temp;
```

وفي حلقة for الأولى يتم اختبار هل قيمة temp أكبر من قيمة العنصر arr[j] فإن كانت كذلك يتم تخصيص قيمة العنصر لـ temp وتخصيص قيمة عداد الحلقة للمتغير k حتى يتم حفظ موقع العنصر , وعند تكرار الحلقة يتم الاختبار والتخصيص السابق , وبانتهاء الحلقة تكون قيمة temp أصغر قيمة في المصفوفة , أي أن هذه الطريقة تتلخص في أنه يتم البحث عن أصغر قيمة في المصفوفة ثم استبدالها بالعنصر الأول واستبدال العنصر الأول بالعنصر الذي يحتوي على هذه القيمة , ثم مثل الشيء مع العنصر التالي وهكذا إلى نهاية الحلقة , وبهذا يتضح لنا لماذا القيمة الابتدائية لعداد الحلقة الداخلية الأولى هي قيمة عداد الحلقة الخارجية , وذلك للتبديل بين العنصر الأول وأصغر عنصر ثم العنصر الثاني وأصغر عنصر ثم .... .

وبعد الخروج من الحلقة الأولى يتم استبدال القيم بين العنصر ذي القيمة الصغرى arr[k] والعنصر الأول في المصفوفة arr[i] .

أما الحلقة الداخلية الثانية فتقوم بطباعة المصفوفة بعد كل عملية ترتيب حتى تبين مالذي يحدث .

المصفوفات ثنائية البعد :

في المصفوفة الأحادية البعد كان يتم الوصول لأي عنصر فيها بدليل واحد , أما في المصفوفة ثنائية البعد فإنه يتم الوصول للعنصر بدليلين الدليل الأول يشير إلى الصف والثاني إلى العمود ,

أي أن المصفوفة ثنائية البعد هي أكثر من صف وكل صف يحتوي على عمود وأقرب مثال لها الجداول .

وعدد عناصر المصفوفة هو عدد الصفوف * عدد الأعمدة .

وتعريف هذا النوع من المصفوفات يأخذ الشكل التالي:

```
type array_name[size of rows][size of columns];
```

والتعريف التالي لمصفوفة ثنائية البعد من النوع float بها ثلاثة صفوف وأربع أعمدة :

```
float array[3][4];
```

ويتم إدخالها عناصرها باستخدام حلقتين الأولى للصفوف والثانية للأعمدة , ويتم الإدخال بإدخال عناصر الصف الأول – أعمدته - أولاً ثم الثاني وهكذا كالتالي :

```
for(int i=0;i<3;i++)
      for(j=0;j<4;j++)
         cin>>array[i][j];
```

وطباعة المصفوفة يتم كالتالي:

```
for(int i=0;i<3;i++)
      {
              for(j=0;j<4;j++)
                cout<<array[i][j];
              cout<<endl;
      }
```

ففي الحلقة الداخلية يتم طباعة عناصر الصف i ثم يتم طباعة سطر جديد ثم طباعة الصف الثاني إلى نهاية المصفوفة .

مثال : طباعة محورة مصفوفة حجمها 4×4 وطباعة حاصل ضرب عناصر قطريها الرئيسي والثانوي:

```
int arr[4][4],i,j;
for(i=0;i<4;i++)
{
   cout<<"Enter elements of row "<<i+1<<": ";
   for(j=0;j<4;j++)
     cin>>arr[i][j];
}
cout<<"\nThe array is :\n\n";
for(i=0;i<4;i++)
{
   for(j=0;j<4;j++)
     cout<<"  "<<arr[i][j];
   cout<<endl;
}
cout<<"\nThe array transpose is :\n\n";
for(i=0;i<4;i++)
{
   for(j=0;j<4;j++)
     cout<<"  "<<arr[j][i];
   cout<<endl;
}
int result=1;
for(i=0;i<4;i++)
   result*=arr[i][i];
cout<<"\nMultiplication product of main diagonal elements is : "
    <<result<<endl;
result=1;
for(i=0,j=3;i<4;i++,j--)
    result*=arr[i][j];
cout<<"\nMultiplication product of secondary diagonal elements is : "
    <<result<<endl;
```

السلاسل النصية :

السلاسل النصية Strings والتي تمثل النصوص مثل الأسماء وغيرها هي عبارة عن مصفوفة حرفية أحادية البعد , فتعريف سلسلة نصية وتخزين الكلمة ++C فيها وطباعتها يتم كالتالي:

```
char s[4]="C++";
cout<<s;
```

وقد تم الإعلان عن المصفوفة بحجم 4 مع أن الكلمة تتكون من 3 أحرف لأن اللغة تنهي السلاسل النصية بحرف نهاية السلسلة '\0' .
ويمكن الإدخال باستخدام cin كالتالي:

```
cin>>s;
```

ولكن الإدخال بهذه الطريقة لا يسمح بإدخال أكثر من كلمة , أي أن الإدخال يتوقف عند أول فراغ , ولإدخال سلاسل نصية تحتوي على فراغات يتم باستخدام الدالة getline(s,i) حيث s هي السلسلة النصية , و i عدد الأحرف المراد إدخالها , وتكتب جملة الإدخال كالتالي:

```
cin.getline(s,i);
```

وهذا الشرح للسلاسل النصية ليس إلا البداية فيها, والشرح الشامل ليس في نطاق هذا الكتاب .